

JavaScript

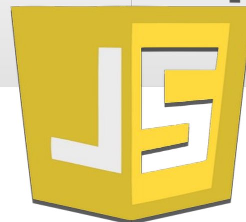
Pr. MAHRAZ Mohamed Adnane



Introduction

JavaScript permet de dynamiser un site Web.

Code JavaScript intégré aux pages HTML.



Code interprété par le navigateur client ≠ code PHP (interprété du coté serveur).

JavaScript est un langage événementiel (association d’actions aux événements déclenchés par l’utilisateur (passage de souris, clic, saisie clavier, etc...).



Intérêts de JavaScript ?

Possibilité de mettre en place des animations sans l'inconvénient des longs temps de chargement nécessités par les données multimédia.

Supporté par les principaux navigateurs, c.-à-d., il ne nécessite pas de plug-in particulier.



Accès aux objets contenus dans un document HTML

Langage relativement sécurisé : il est impossible de lire ou d'écrire sur le disque client (impossibilité de récupérer un virus par ce biais).



Intégration de JavaScript dans HTML

Il existe **2 manières** pour insérer un code JavaScript dans une page HTML:

JavaScript dans HTML

```
<html>
    <head>
        <title>Page HTML</title>
    </head>
    <body>
        <script>
            alert('bonjour');
        </script>
    </body>
</html>
```

JavaScript à l'extérieur du HTML

```
<html>
    <head>
        <title>Page HTML</title>
        <script src="monScript.js">
        </script>
    </head>
    <body>
    </body>
</html>
```



Entrée et sortie de données avec JS

Fichier Édition Affichage Historique Marque-pages Outils ?

Exemple d'un site web

file:///D:/mon cours Web dynamique/seance6/exemples_cours/index.html

Rechercher

c'est un texte affiché par la méthode write de l'objet Document



Entrée et sortie de données avec JS

```
<html>
   <head>
      <title> une page simple </title>
   </head>
   <body>
      Bonjour
      <script>
         alert('bonjour');
         document.write (prompt('quel est votre nom ?','Indiquer votre nom ici'));
         confirm('quel bouton allez-vous choisir ?');
      </script>
   </body>
</html>
```



Fonctions simples

Syntaxe:

```
function nom_fonction ([param1, …]){
    //Corps de la fonction
}
```

Corps de la fonction :

Déclaration des variables,
Instructions réalisés par la fonction,
Instruction **return** pour renvoyer une valeur ou un objet (Facultative)

JavaScript



Déclaration de variables

4 façons pour déclarer une variable en JavaScript :

- En utilisant le mot clé ***va***
- En utilisant le mot clé
- En utilisant le mot clé ***co***
- Ne rien utiliser.

- Le mot clé **const** permet de déclarer une variable dont le contenu est inchangeable.



Déclaration de variables

- Utilisation de l’instruction

  [var , let , const ,   ] variable=valeur;

  - Pas de typage (détection automatique par l’interpréteur)
  - Nom de variable **sensible à la casse**.
  - Portée :
    - **globale** ⇒ visibilité dans tout le module
    - **fonction** ⇒ visibilité dans toute la fonction
    - **bloc** ⇒ visibilité dans le bloc interne



Portée fonction et portée globale

- Une variable déclarée avec le mot clé **var** pourra avoir deux portées: **globale** et **fonction**.
- La portée d'une variable déclarée par **var** dépend de l'endroit où elle est déclarée :
  - Portée globale: déclarée en dehors de la fonction.
  - Portée fonction: déclarée à l'intérieur d'une fonction aura une portée limitée à cette seule fonction.

```
var nom="Mohamed"; //variable globale
function afficher(){
    console.log("Votre nom est: "+nom);
}
console.log("Votre nom est: "+nom);
```

```
var nom="Mohamed"; //variable globale
function saisir(){
    var nom; //variable locale
    nom=prompt("Quel est votre nom:");
    console.log(window.nom);
    return nom;
}
console.log("Votre nom est: "+saisir());
console.log("Votre nom est: "+nom);
```



Déclaration de variables: portée bloc

- Avant ES6 (2015), JavaScript n'avait que la **portée globale** et la **portée fonction**.
- ES6 a introduit deux nouveaux mots-clés JavaScript importants : **let** et **const**.
- Ces deux mots-clés fournissent **une portée bloc**.
- Les variables déclarées à **l'intérieur d'un bloc { }** ne sont pas accessibles depuis **l'extérieur du bloc**



Déclaration de variables: portée bloc

Variable *nom* à une portée bloc

```
function test() {
    if (true) {
        let nom = "Rachid";
    }

console.log(nom);/*ReferenceError
                  :nom    is    not
defined*/
}
```

Variable *nom* à une portée fonction

```
function test() {
    if (true) {
        var nom = "Rachid";
    }
    console.log(nom); // Rachid
}
```



Déclaration de variables: portée bloc

La portée bloc signifie que vous pouvez masquer des variables dans une fonction :

```
function test() {
  let nb = 5;
  if (···) {
     let nb = 10; // Masquer la variable externe `nb`
     console.log(nb); // 10
  }
  console.log(nb); // 5
}
```



Déclaration de variables: let vs const

- Une variable déclarée avec **let** est modifiable :

```
let mot = "abcd";
mot = "Rachid";
console.log(mot); // Rachid
```

- Une variable déclarée avec **const** est immuable :

```
const mot = "abcd";
mot = ‘Rachid'; // TypeError
```



Structures de contrôle

- Test conditionnel : if … else …

```
<script>
    let age=prompt("quel est votre age");
    if (age>=18){
              alert("vous etes Majeur...");
    }else{
              alert("vous etes Mineur...");
    }
</script>
```



Structures de contrôle

- Boucle itérative :

for(initialisation ; condition ; opération) { ... instructions ... }

```
<script>
    let nb=prompt("Donnez un nombre");
    let somme=0;
    for(let i=1;i<=nb;i++){
              somme+=i;
    }
    alert("La somme des nombres entre 0 est "+ nb +" est "+somme);
</script>
```



Structures de contrôle

- Boucle conditionnelle

while(condition) { … instructions … }

```
<script>
    let nb=prompt("Donnez un nombre");
    let somme=0, i=0;
    while(i<=nb){
                somme+=i;
                i++;
    }
    alert("la somme des nombres entre 0 est "+nb+" est "+somme);
</script>
```



Expressions de Fonction

syntaxe :

const nom_fonction= function ([param1, …]){
//Corps de la fonction
}

const nom_fonction=([param1, …])=>{
//Corps de la fonction
}

```
<script>
    const add=function(x,y){
          console.log("La   somme   est:"+(x+y));
    }
</script>
```

```
<script>
    const add= (x,y)=>{
          console.log("La   somme   est:"+(x+y));
    }
</script>
```



Expressions de Fonction et Portée

const nom_fonction=([param1, …])=>{
//Corps de la fonction
}

- ***const*** et ***let*** permet d'empêcher la redéfinition de la fonction une deuxième fois puisqu'ils sont de type ***bloc***.
- Si on a une seule instruction dans une fonction, on peut ne pas utiliser le mot clé **return** ni d'accolades **{ }.**

```
const add = (x, y) => x + y;
console.log(add(1, 2)); // 3
//On essaie de changer la fonction
add = (x, y) => x - y; // Uncaught TypeError: Assignment to constant variable.
```



Fonctions anonymes

Pour que cette fonction s’appelle automatiquement:

(()=>{//Corps de la fonction})();

- Pas de nom pour la fonction
- **=>** Isoler les variables déclarées au sein de la fonction au monde extérieur.

```
(()=>{
    alert("Fonction anonyme appelée automatiquement");
})();
```



JavaScript

Objets prédéfinis



Objets prédéfinis

- Plusieurs objets prédéfinis en JavaScript:
  - Array, Boolean, Date, Function, Image, Number, Object, ou String.
- L’opérateur Typeof
  - L'opérateur typeof renvoie une chaîne de caractères indiquant quel est le type de l'opérande.

```
let titre="Les raisins de la colère";
typeof titre; //retourne string
```

```
function message() {
    console.log("Bonjour!");
}

console.log(typeof message);                // => function
console.log(message instanceof Object);     // => true
```



Tableau de données (Objet array)

- Déclaration par l’utilisation de var/let.
- Le premier élément du tableau est indexé à 0.
- Il est possible de déclarer un tableau sans dimension fixée: Sa taille s'adapte en fonction du contenu.

```
   // création implicite d’un tableau
let mon_tableau = ["Ali", ‘Mohamed', "Sarah", 10,
6];

   // création d’un tableau de 10 éléments
let mon_tableau = Array(10);

   // création d’un tableau avec l’opérateur
« new »
let mon_tableau = new Array(10);
```



Utilisation de tableaux

- Accès aux éléments d'un tableau: Utilisation des crochets : [   ]

```
   let tableau=new Array();
   tableau[0]=10;
   tableau[1]=5;
```

- La proprièté Length

```
tableau.length
```

- Parcourir un tableau

```
   // Parcourir un tableau sans connaître le nombre d'éléments
   let tableau= new Array(1, "a", 9) ;
   for (let i=0; i<tableau.length; i++)
        console.log("tableau[" + i + "] = "+tableau[i]);
```



Utilisation de tableaux

- Parcourir un tableau sans connaitre le nombre d’éléments

```
let tableau= new Array(1, "a", 9) ;
tableau[200] = 12 ;
for (let i in tableau)
      console.log("tableau[" + i + "] = "+tableau[i]);
```

- Parcourir un tableau sans connaitre le nombre d’éléments avec ES6

```
let tableau = new Array(1, "a", 9) ;
tableau.forEach( (data,index)=> {
                 console.log("tab["+index+"]="+data);
});
```



Tableaux associatifs

- L’indice est une chaîne de caractères

```
   let tab=new Array();
   tab["nom"]  ="Ben ali";
   tab["prenom"]   ="Mohamed";
   tab["age"]  =25;
   tab["adresse"] ="Fes";
   ...
```

```
   ...
   alert("Votre nom est: "+tab["nom"]);
...
```

La propriété **length** de l’objet **Array**() pour ce genre de tableau ne fonctionne pas.



L'objet Array

Il y a des méthodes pour manipuler l’objet **Array** :

- *concat( )* : permet de concaténer 2 tableaux;
- *join( )* : converti un tableau en chaîne de caractères;
- *slice( )* : retourne une section du tableau;
- *sort( )* : permet le classement des éléments du tableau;
- *reverse( )* : inverse le classement des éléments du tableau;



L'objet String

- L'objet String permet de manipuler les chaînes de caractères
- Propriété :
  - length : retourne la longueur de la chaîne de caractères;
- Méthodes :
- Opérations sur les chaînes
  - concat(str) : retourne la chaîne concaténée avec str
  - split(str) : retourne, sous forme de tableau, les portions de la chaînes délimitées par str
  - substring(debut,fin) : extrait une sous-chaîne, depuis la position debut (incluse) à fin (excluse).
  - substr(debut,i) : extrait une sous-chaîne, depuis la position debut, en prenant i caractères



L'objet String

- Opérations sur les caractères
  - charAt(i) : retourne le iième caractère
  - indexOf(str) : retourne la position de str dans la chaîne (-1 si elle n'est pas trouvée)
  - lastIndexOf(str) : idem, mais renvoie la position de la dernière occurrence de **str**
  - toLowerCase() : retourne la chaîne en minuscules
  - toUpperCase() : retourne la chaîne en majuscules



L'objet Math

- Propriétés :
  - E : renvoie la valeur de la constante d'Euler (~2.718);
  - LN2 : renvoie le logarithme népérien de 2 (~0.693);
  - LN10 : renvoie le logarithme népérien de 10 (~2.302);
  - LOG2E : renvoie le logarithme en base 2 de e (~1.442);
  - LOG10E : renvoie le logarithme en base 10 de e (~0.434);
  - PI : renvoie la valeur du nombre pi (~3.14159);
  - SQRT1_2 : renvoie 1 sur racine carrée de 2 (~0.707);
  - SQRT2 : renvoie la racine carrée de 2 (~1.414);



L'objet Math

- Méthodes :
  - abs( ), exp( ), log(), sin( ), cos( ), tan( ), asin( ), acos( ), atan( ), max( ), min( ), sqrt( ) sont les opérations mathématiques habituelles;
  - atan2( ) : retourne la valeur radian de l'angle entre l'axe des abscisses et un point;
  - ceil( ) : retourne le plus petit entier supérieur à un nombre;
  - floor( ) : retourne le plus grand entier inférieur à un nombre;
  - pow( ) : retourne le résultat d'un nombre mis à une certaine puissance;
  - random( ) : retourne un nombre aléatoire entre 0 et 1;



L'objet Images

- Propriétés
  - complete
  - width
  - height
  - src
- Méthodes
  - constructeur
    - Image()
    - Image(largeur, hauteur)

```
// Exemple
img = new Image() ;
img.src = 'image.gif' ; // Préchargement
img.onload = function(){
   // Modification de la 13e image de la
page Web
    document.images[12].src = img.src ;
}
```



L'objet Date

- Méthodes
  - Constructeur
  - getDay(), attention de 0 (dimanche) à 6 (samedi)…
  - getDate() / setDate()
  - getMonth() / setMonth(), attention de 0 à 11…
  - getYear() / setYear() / getFullYear() / setFullYear()
  - getHours()/ setHours()
  - getMinutes() / setMinutes()
  - getTime() / setTime()

```
let jour = new Date();
alert(jour.getFullYear());
// 2022
let anniversaire= new Date(2022, 10, 25);
alert(anniversaire.toLocaleString());
// lundi 25 octobre 2022 00:00
```



L’objet Function

- Une fonction est bien un objet.
- Les fonctions JavaScript sont un type spécial d'objets, appelés objets de fonction.
- Un objet fonction comprend une chaîne qui contient le code réel de la fonction.
- Bien que cela ne soit pas recommandé, vous pouvez créer un nouvel objet fonction en transmettant au constructeur **Function** le nom et le corps de la fonction.

```
var body = "return Math.PI * radius * radius";
var circle = new Function("radius", body);

console.log(circle(5));          // => 78.5398..
```



Fonctions supérieures

- eval(chaine)
- isFinite(nombre)
- isNaN(objet)
- parseFloat(chaine)
- parseInt(chaine)



Fonctions supérieures

document.write(isFinite(Math.log(0))) ;

□ false

document.write(isNaN("abcd")) ;

□ true

document.write("12.34"+2) ;

□ 12.342

document.write(parseFloat("12.34")+2) ;

□ 14.34



JavaScript

Objets propres



Déclaration et création d’objets: Objets propres

- Création d’objets avec des initialiseurs d'objets (objets littéraux).

```
const obj= {
  propriété_1: valeur_1,
  propriété_2: valeur_2,
  …
  propriété_n: valeur_n
};
```

```
const obj = new Object();
obj.popriété_1=valeur_1;
obj.propriété_2=valeur_2,
  …
obj.propriété_n= valeur_n;
```

- Les objets sont créés de la même façon qu'avec new Object().
- les objets créés à partir d'une expression littérale seront des instances d'Object



Déclaration et création d’objets: Objets propres

- Les objets peuvent également être créés en utilisant la méthode Object.create().

```
const Animal = {
   type: "Invertébrés", // Valeur par défaut
   afficherType : function() { // Une méthode pour afficher le type Animal
   console.log(this.type);
   }
}
let animal1 = Object.create(Animal); // On crée un nouveau type d'animal:
animal1 animal1.afficherType(); // affichera Invertébrés let animal2 =
Object.create(Animal); // On crée un nouveau type d'animal: animal2
animal2["type"]= "poisson"; // ou bien animal2.type= "poisson";
animal2.afficherType(); // affichera poisson
```



Déclaration et création d’objets: Objets propres

Création d’objets propres en utilisant un constructeur

- Par appel d’une fonction qui va créer les propriétés de l’objet.
- Utilisation de **this** pour faire référence à l’objet courant
- On crée une instance de l'objet avec **new**.

```
function Etudiant(Le_nom,Le_prenom,Le_CODE){
             this.nom=Le_nom;
             this.prenom=Le_prenom;
             this.Code=Le_CODE;
}
let e1=new Etudiant("Mohamed", "Ben Ali", "1298742046");
alert("Votre nom est: "+e1.prenom);
```



Déclaration et création d'objets: Objets propres

- Déclaration de méthodes
  - Association de fonctions dans la création de l'objet.

```
function Etudiant(Le_nom,Le_prenom,Le_CODE){
             this.nom=Le_nom;
             this.prenom=Le_prenom;
             this.CODE=Le_CODE;
             this.afficher=affiche_Etudiant;
}
function affiche_Etudiant(){
             console.log("Votre nom et prénom est: "+ this.nom+"
"               +this.prenom+",Votre CODE est: "+ this.CODE );
}
let e=new Etudiant("Mohamed", "Ben Ali", "1298742046");
e.afficher();
```



Exercice

1. Définir une fonction constructeur « Produit » avec les attributs libelle , catégorie et prix et une méthode description() pour afficher le détail du produit.
2. Définir une fonction constructeur « Commande » qui comporte un attribut tableau de produits, une méthode ajouter() permettant d'ajouter un produit au tableau et une méthode affiche() permettant d'afficher le tableau de produits.
3. Créer une instance de « Commande » et y ajouter des produits.
4. Afficher les produits d'une commande en utilisant la méthode affiche().



JavaScript

Programmation événementielle



Déclenchement d’instructions JavaScript

- Programmation événementielle
  - JavaScript = langage réactif
  - L’interaction avec l’utilisateur est gérée via des événements
  - Événement = tout changement d’état du navigateur



Déclenchement d’instructions JavaScript

- Événements JavaScript
  - blur : le focus est enlevé d’un objet
  - focus : le focus est donné à un objet
  - change : la valeur d’un champ de formulaire à été modifiée par l’utilisateur
  - mouseover : la souris est déplacée sur un objet
  - click : un clic souris est déclenché sur un objet
  - select : un champ de formulaire est sélectionné (par tabulation)
  - submit : un formulaire est soumis
  - load : la page est chargée par le navigateur
  - unload : l’utilisateur quitte la page



Déclenchement d’instructions JavaScript

Il est possible de baser l’exécution de fonctions sur des événements

Événements détectables

- Nom de l’événement précédé de **on** :
  onBlur, onChange, onClick, onFocus, onLoad, onMouseover, onSelect, onSubmit, onUnload

Association événement - action

- Dans le code HTML, identique à la déclaration d ’une propriété :
- <nom_élément attribut$_i$ = propriété$_i$ événement$_j$ = "action$_j$" >
- <button onClick=" envoyer()">Envoyer<button>



Déclenchement d'instructions JavaScript

```
<html>
     <head>
           <title>Exemples de déclenchements</title>
           <script>
                        function saluer() {
                             alert("Bonjour tout le monde");
                        }
           </script>
     </head>
     <body onLoad="saluer()">
           <h1 onMouseover="saluer()">Survoler le pointeur pour exécuter l'événement</h1>
           <form>
                <input type="button" name="bouton"  value="salut" onClick="saluer()">
           </form>
           <h1>Exécution sur protocole javascript:</h1>
           <a href="javascript:saluer()">pour saluer</a>
     </body>
</html>
```



Changer l’aspect du formulaire

Ecrire une page HTML contenant un formulaire (deux zones de texte et le bouton envoyer). La bordure de la zone du texte est changée en vert s’elle est sélectionnée, sinon, elle devient en gris.



Changer l’aspect du formulaire

```
<form>
   <input type="text" value="" name="texte1" onBlur="unchanger(this)" onFocus="changer(this)"/>
   <input type="text" value="" name="texte2" onBlur="unchanger(this)" onFocus
="changer(this)"/>
   <input type="submit"/>
</form>
<script>
    let changer=function(texte){
    texte.style.border="2px solid green";
    }
    let unchanger=function(texte){
     texte.style.border="";
    }
</script>
```



Contrôle du formulaire

Ecrire une page HTML contenant un formulaire (zone de texte et le bouton envoyer). Un message d’erreur est affiché si la zone de texte est vide au moment de la soumission.



Contrôle du formulaire

```
<form onSubmit="return verifier()">
   <input type="text" name="texte" value="" name="texte1" />
   <input type="submit" />
</form>
<script>
   let verifier=function(){
      if (document.forms[0].elements["texte"].value==""){
         alert("zone vide");
         return false;
      }
   }
</script>
```



DOM

Document Object Model



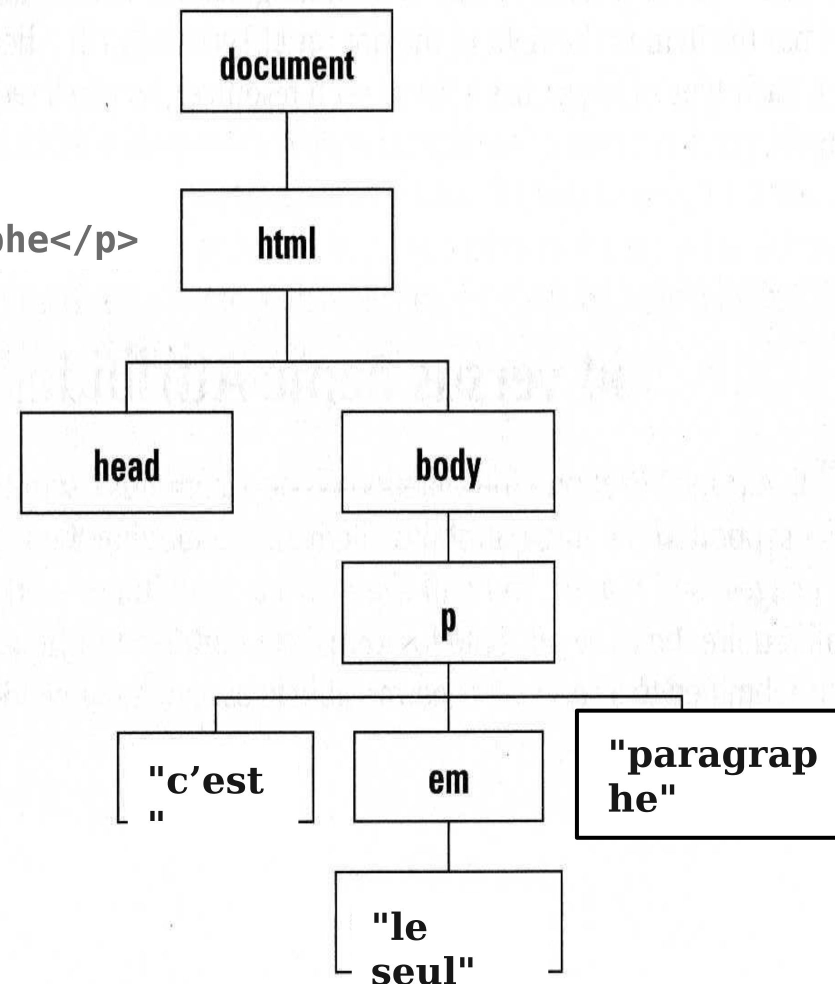
DOM = Document Object Model

- Le modèle d'objet du document (DOM) donne une représentation en mémoire des objets du document.
- Un objet est un élément HTML.
- Le DOM est l'adresse par laquelle vous pouvez localiser un objet de la page HTML.
- Un objet peut être récupéré et exploité par le Javascript (ou un autre langage de programmation).



DOM = Document Object Model

- Le DOM décrit le chemin partant de la fenêtre du navigateur pour descendre jusqu'aux objet de la page Web.
- Le DOM est structuré comme un arbre est suit de près la structure hiérarchique du code HTML.
- L'arbre contient des nœuds, les nœuds peuvent avoir des fils, et tous les nœuds ont un parent (sauf la racine).





Utilisation du DOM

A l’aide de Javascript :

- On peut sélectionner un élément (<p> par exemple), et modifier sa couleur (DOM document + DOM element).
- On peut sélectionner un élément et lui assigner un événement (DOM document + DOM events).
- On peut sélectionner les attributs ("title" par exemple) et changer leur contenu (je remplace title="image2" par title="beau tigre") (DOM document + DOM attribute).



DOM « document »

- Le DOM « document » qui permet de sélectionner un objet au sien d’un document:
  - document.getElementById() ; // par son ID
  - document.getElementsByName() ; // par son attribut « name »
  - document.getElementsByTagName() ; // par son nom de balise HTML
  - document.getElementsByClassName() ; // par son nom de la classe

```
<input type="search" class="c1" name="rechercher" id="rechercher"/>
```

```
let zone1=document.getElementById("rechercher");
let
zone2=document.getElementsByName("rechercher");
let zone3=document.getElementsByTagName("input");
```



DOM « document »

- Deux autres méthodes basées sur les selecteurs CSS:
  - **querySelectorAll(selector):** retourne tous les éléments correspondant au **selector css.**
  - **querySelector(selector):** retourne uniquement le premier élément trouvé.

```
<input type="text" class="c1" name="text" id="text"/>
<input type="search" class=" c1" name="recherche" id="rechercher"/>
```

```
document.querySelectorAll(". c1")[1]; //retourne le deuxième élément qui a la classe «  c1 ».
let zone2=document.querySelector("#text"); //retourne le premier élément qui a un id «  text  ».
```



DOM « element » +DOM « attribute »

- Le DOM « element » permet de faire une action sur les éléments sélectionnés.
- Le DOM « attribute» permet de de modifier les attributs des éléments sélectionnés

```
<input type="search" name="recherche" id="rechercher"/>
```

```
let zone1=document.getElementById("rechercher"); // selectionne la zone de recherche
zone1.style.color="red"; //change la couleur du texte
Zone1.style.setProperty('color','red');   //changer une propriété
Var attribut=zone1.getAttribute("type"); // renvoie « search »
zone1.setAttribute("placeholder", " Texte à chercher " ); // ajout l'attribut « placeholder »
```



DOM « events »

- Le DOM « events » permet de faire une action lors d'un événement (exemple au clic de la souris)

```
<input type="search" placeholder="Texte à chercher" name="recherche"
id="c"/>
```

```
<script>
    let
zone=document.getElementById("c");
    zone.onfocus=function(){
    zone.value="";
    }
</script>
```

```
<script>
    let
zone=document.getElementById("c");
    zone.onfocus=vider;
    function vider(){
    zone.value="";
    }
</script>
```



DOM « events »: addEventListener

La méthode **addEventListener** permet d’abonner à l’objet sur lequel elle est invoquée une fonction pour l’événement précisé.

objet.addEventListener(eventType, listenerFunction)

- objet : l’objet ciblé (window, document ou un élément de la page).
- eventType : une chaîne de caractères désignant le type d’événement:

  "**click**", "**load**", "**change**", "**mouseover**", "**keypress**" etc.
- listenerFunction : la fonction listener qui sera appelée lorsque l’événement se produit



DOM « events »: addEventListener

- click
- dblclick
- mousedown
- mousemove
- mouseover
- mouseout
- mouseup
- keydown
- keypress
- keyup
- abort
- Error
- load
- resize
- scroll
- unload
- blur
- change
- focus
- reset
- select
- submit



DOM « events »: addEventListener

- **Exemple 1:**

```
window.addEventListener('load',function(){
    console.log('la page est totalement chargée');
});
```

- **Exemple 2:**

```
document.addEventListener('DOMContentLoaded',function(){
    let img = document.getElementsByTagName("img")[0];
    img.addEventListener('mouseover',function(){
        img.style.opacity=0.5;
    });
    img.addEventListener('mouseout',function(){
        img.style.setProperty('opacity',1);
    });
});
```



DOM « events »: addEventListener (listener Function)

- dans une fonction **listener**, la variable **this** est définie et désigne l’objet qui a déclenché l’événement typiquement l’élément du document.
- un objet **event** est créé pour chaque événement. Cet objet est passé en paramètre lors du déclenchement de la fonction **listener** associée. Le type d’objet **event** varie selon l’événement. Un objet **event** possède des propriétés qui informent sur l’événement.



DOM « events »: addEventListener

```
document.addEventListener("DOMContentLoaded", () => {
  let images = document.querySelectorAll("img");
  images.forEach((img) => {
    img.addEventListener("mousemove", (event) => {
event.target.style.setProperty("opacity", 0.5);
    });
  });
  images.forEach((img) => {
    img.addEventListener("mouseout", (event) => {
      event.target.style.setProperty("opacity", 1);
    });
  });
});
```

Event interface

Event interface



Modifier le DOM

Deux Solutions pour modifier le DOM:

- **innerHTML**
- **DOM « pur »**



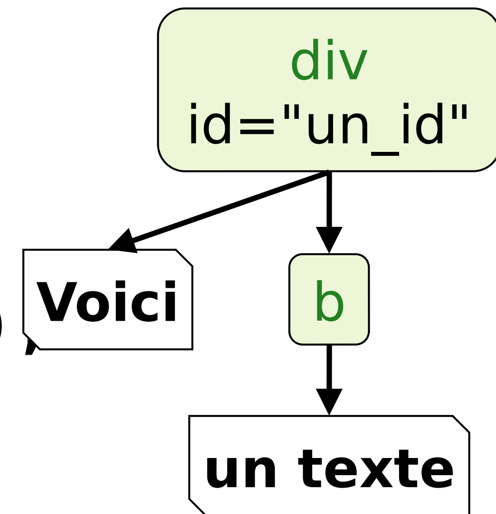
Modifier le DOM: innerHTML

- Identifier un élément HTML
```
<div id="un_id"></div>
```
- Accéder à un élément
```
e = document.getElementById("un_id");
```
- Construire une chaîne contenant du HTML
```
s = "Voici <b>un texte</b>";
```
- Modifier le contenu de l’élément
```
e.innerHTML = s;
```
- Interprétation « automatique » par le navigateur du nouveau contenu pour modifier le document



Modifier le DOM « pur »

- Identifier un élément HTML
```
<div id="un_id"></div>
```
- Accéder à un élément
```
e = document.getElementById("un_id");
```
- Créer un nœud de type « texte »
```
t1 = document.createTextNode('Voici ');
t2 = document.createTextNode('un texte');
```
- Créer un nouveau nœud de type « balise »
```
b = document.createElement('b');
```
- Construire des liens de parenté
```
e.appendChild(t1);
b.appendChild(t2); e.appendChild(b);
```





Exercice

- Ecrire une page HTML qui contient un script qui vérifie lors de la soumission du formulaire, si le champ n’est pas vide, sinon, un message d’erreur est affiché.
  1. Utiliser innerHTML pour insérer le message dans la balise span, en gras avec une couleur rouge.
  2. Utiliser une modification manuelle sur le DOM.

Veuillez remplir tous les champs!

Mahraz

S'inscrire

//classe bootstrap form-control is-valid

//classe bootstrap form-control is-invalid



Solution: HTML

```
<body>
    <div class="container">
        <form>
            <div id="message" class="alert alert-danger"
role="alert">
                Veuillez remplir tous les champs!
            </div>
            <input type="text" name="nom" class="form-control"
id="firstname"/>
            <input type="text" name="prenom" class="form-control"
id="lastname"/>
            <input type="button" class="btn btn-outline-primary"
value="S'inscrire">
        </form>
    </div>

</body>
```



Solution : script

```
<script>

document.addEventListener("DOMContentLoaded",function(){
let message=document.getElementById("message");
message.style.display="none"
let btn=document.querySelector("input[type=button]");
btn.addEventListener("click",function(){
 let text1=document.querySelectorAll("input[type=text]");
 if (text1[0].value==""){
  text1[0].setAttribute("class","form-control is-
invalid");
 }else{
  text1[0].setAttribute("class","form-control is-valid");
}
if (text1[1].value==""){
 text1[1].setAttribute("class","form-control is-
invalid");
}
 else{
                text1[1].setAttribute("class","form-
control is-valid");
     } message.style.setProperty("display","block")
 if (text1[0].getAttribute("class")=="form-control is-
invalid" || text1[1].getAttribute("class")=="form-
control is-invalid"){
     message.setAttribute("class","alert alert-
danger")
     message.innerHTML="<b>Veuillez remplir tous les
champs!</b>";
  }else{
   message.setAttribute("class","alert alert-info")
   message.innerHTML="<b>Données envoyées avec
succès</b>";
  }
 })
})
</script>
```



JavaScript

Cookie



Cookies

- **Un *Cookie* est une chaîne de caractères qu'une page HTML (contenant du code JavaScript) peut écrire à un emplacement UNIQUE et bien défini sur le disque dur du client.**
  - Cette chaîne de caractères ne peut être lue que par le seul serveur qui l'a générée.
- **Que faire avec un cookie**
  - Transmettre des valeurs (contenu de variables) d'une page HTML à une autre.
    - Par exemple, créer un site marchand et constituer un "caddie" pour le client. Caddie qui restera sur son poste et vous permettra d'évaluer la facture finale au bout de la commande. Sans faire appel à quelque serveur que ce soit.
  - Personnaliser les pages présentées à l'utilisateur en reprenant par exemple son nom en haut de chaque page.



Cookies: Limitations

- On ne peut pas écrire autant de cookies que l'on veut sur le poste de l'utilisateur (client d’une page).
- Il y a des limites :
  - Limites en nombre : Un seul serveur (ou domaine) ne peut pas être autorisé à écrire plus de 20 cookies.
  - Limites en taille : un cookie ne peut excéder 4 Ko.
  - Limites du poste client : Un poste client ne peut stocker plus de 300 cookies en tout.



Cookies: Structure

Nom=Contenu; expires=expdate; path=Chemin; domain=NomDeDomaine; secure

- Nom=Contenu;
  - Sont deux variables suivies d'un ";" . Elles représentent l'en-tête du cookie.
  - La variable *Nom* contient le nom à donner au cookie.
  - La letiable *Contenu* contient le contenu du cookie
  - Exemple ma_cookie=« oui:visite»



Cookies: Structure

- **Expires= expdate;**
  - Le mot réservé expires suivi du signe "=" (égal). Derrière ce signe, vous mettrez une date d'expiration représentant la date à laquelle le cookie sera supprimé du disque dur du client.
  - La date d’expiration doit être au format :
    Wdy, DD-Mon-YYYY HH:MM:SS GMT
  - Utiliser les fonctions de l'objet Date
  - Règle générale : 'indiquer un délai en nombre de jours (ou d'années) avant disparition du Cookie.



Cookies: Structure

- **path=Chemin;**
  - path représente le chemin de la page qui a créé le cookie.
- **domain=NomDeDomaine;**
  - domain représente le nom du domaine de cette même page
- **secure**
  - secure prend les valeurs "true" ou "false" : Il permet de spécifier que le *cookie* sera envoyé uniquement si la connexion est sécurisée selon que le cookie doit utiliser des protocoles HTTP simples (non sécurisés) ou HTTPS (sécurisés).
- Les arguments path, domain et secure sont facultatifs.
  - lorsque ces arguments sont omis, les valeurs par défaut sont prises.
  - Pour secure, la valeur est "False" par défaut.



Cookies: Mode d'écriture

- Un cookie est une propriété de l'objet document (la page HTML chargée dans le navigateur) alors l'instruction d'écriture de cookie est:

document.cookie = Nom + "=" + Contenu + "; expires=" + expdate.toUTCString() ;

```
let Nom = "MonCookie" ; // nom du cookie
let Contenu = "Hé... Vous avez un cookie sur votre disque !" ; // contenu du cookie
let expdate = new Date () ; // crée un objet date indispensable
puis rajoutons lui 10 jours d'existence :
expdate.setTime (expdate.getTime() + ( 10 * 24 * 60 * 60 * 1000)) ;
document.cookie = Nom + "=" + Contenu + "; expires=" + expdate.toUTCString() ;
```



Cookies: Mode lecture/Modification

- **Lecture d'un cookie**

```
let LesCookies ; // pour voir les cookies
LesCookies = document.cookie ; // on met les cookies dans la variable LesCookies
```

  - Accéder à la propriété cookie de l'objet document.
  - **Document.cookie**

- **Modification d'un cookie**
  - Modifier le contenu de la variable *Contenu* puis réécrire le cookie sur le disque dur du client

```
Contenu = "Le cookie a été modifié..." ; // nouveau contenu
document.cookie = Nom + "=" + Contenu + "; expires=" + expdate.toGMTString() ; // écriture sur le disque
```



Cookies: Suppression

- Positionner la date de péremption du cookie à une valeur inférieure à celle du moment où on l'écrit sur le disque.

```
// on enlève une seconde (ça suffit mais c'est nécessaire)
expdate.setTime (expdate.getTime() - (1000)) ;

// écriture sur le disque
document.cookie = Nom + "=" + Contenu + "; expires=" + expdate.toGMTString()
```



JavaScript

Expressions régulières



Expressions régulières

let exp = new **RegExp**('modèle'[,'options']);
let exp = /modèle/[options]

- Options :
  - i insensible à la casse
  - m multi-ligne
  - g toutes les occurrences

Méthodes

- exp.test(ch) true si correspondance
- exp.exec(ch) première occurrence capturées
- ch.search(exp) indice 1ere occurrence
- ch.match(exp) tableau de correspondances et les groupes mémorisés
- ch.replace(exp, ch2) remplace les occurrences par ch2

```
let exp = /(\d{2}) (\w+) (\d{4})/;
let ch = "28 octobre 2019";
if (exp.test(ch)){
alert(ch.replace(exp, "annee : $3, mois : $2, jour : $1"));
// annee : 2019, mois : octobre, jour : 28
}
```



Expressions régulières

Indicateurs d'occurrence

{n} exactement n fois * {0,}
{n,} au moins n fois + {1,}
{n,m} entre n et m fois ? {0,1}

Caractères spéciaux

| ou . tout caractère sauf \n
\t tabulation \n saut de ligne
\0 car. nul \ car. d'échappement

Classes de caractères

[abc] un caractère parmi a, b ou c
[a-z] intervalle : un caractère de a à z
[^ab] un caractère autre que a ou b
\d un chiffre \D tout sauf un chiffre
\w [a-zA-Z0-9_] \W tout sauf mot
\s espacement \S tout sauf car. esp.

Correspondances dans la chaîne

^ début $ fin

Mémorisation

(x) Mémoriser sous expression x



Expressions régulières: Exemples

```
  <script type="text/javascript">
true   document.write(/l/.test('Hello')) ;
false  document.write(/^l/.test('Hello')) ;
false  document.write(/^h/.test('Hello')) ;
true   document.write(/^h/i.test('Hello')) ;
true   document.write(/^Hel.o/.test('Hello')) ;
true   document.write(/^Hel+o/.test('Hello')) ;
true   document.write(/^He+llo/.test('Hello')) ;
true   document.write(/^Hea*llo$/.test('Hello')) ;
true   document.write(/^He(l|o)*$/.test('Hello')) ;
true   document.write(/^H[leos]+/.test('Hello')) ;
false  document.write(/^H[^leo]+/.test('Hello')) ;
true   document.write(/^H[^kyz]+/.test('Hello')) ;
true   document.write(/^H[a-z]*/.test('Hello')) ;
true   document.write(/^H[a-z]*$/.test('Hello')) ;
  </script>
```



Expressions régulières: Exercice

- Ecrire une fonction en JS permettant d’extraire toutes les balises HTML contenant dans une zone de texte. La fonction doit renvoyer un tableau des balises trouvées.



Expressions régulières: Solution

```
<textarea id="txt" rows="4"></textarea>
<button id="bt">Tester</button> <script>
    function chercherbalise(html){
        let reg=new RegExp('<[^>/]+/?>','gi');
        let balises=new Array();
        balises=html.match(reg);
        return balises;
    }
    let bt=document.getElementById("bt");
    bt.onclick=function () {
        let txt=document.getElementById("txt");
        console.log(chercherbalise(txt.value))
    }
</script>
```



JQuery

write less do more



Introduction

- Une bibliothèque ou une API (Application Programming Interface) Javascript.
- Elle permet de manipuler très facilement l'arbre DOM à l'aide d'une syntaxe simplifiée.
- JQuery permet de changer/ajouter une classe CSS, créer des animations, modifier des attributs, etc.

```
Javascript : document.getElementsByTagName("LI")
[0].innerHTML="Lait";
JQuery : $("LI:first").html("Lait");
```



une simple bibliothèque à importer

- Disponible sur le site de Jquery: http://jquery.com/

```
<script type="text/javascript" src="jquery.js"></script>
```

- Ou directement sur Google code

```
<script type="text/javascript"
 src="http://ajax.googleapis.com/ajax/libs/jquery/1/jque
 ry.min.js">
</script>
```



La fonction jQuery()

- Le signe « $ » de JQuery remplace absolument querySelector() et querySelectorAll() du DOM en Javascript.
- Il permet à lui seul de sélectionner n’importe quel « composite » de notre document HTML (balises, attributs, ID, classes, pseudo-éléments etc …)



Sélecteur magique : $('sélecteur') !

- $("*") **// permet de sélectionner tous les éléments HTML**
- $("h1") **// sélectionne toutes les balises « h1 »**
- $("#nom-de-ID") **// permet de sélectionner les éléments par leur « id »**
- $(".nom-de-la-classe") **// par leur classe**
- $("h1, a, #nom-id, .nom-classe, div,p") **// plusieurs éléments mélangés (classes, id, nom)**
- $(":text") **// tous les types « text »**
- $("img[alt]") **// tous les « img » ayant l'attribut « alt »**
- $("[alt]") **// tous les attributs « alt »**
- $("img[title='tigre']") **// tous les « img » ayant l'attribut « alt » qui porte la valeur « tigre »**



$(sélecteur)

- $ accepte des sélecteurs spécifiques :
  - $(':first'), $(':last'), $(':header')
- des sélecteurs en forme de filtres :
  - $(':even'), $(':odd'), $(':visible'), $(':hidden')
  - plus fort: $(':contains(du texte)')
- des attributs
  - $('a[href]'), $('a[href^=http://]'), $('img[src$=.png]')



Fonctions pour CSS

- $("a").css("display","block"); **// pour écrire une propriété CSS**
- $("label").css({"float":"left", "font-style":"italic", "width":"100px" }); **// pour écrire plusieurs propriétés CSS**
- $("h1").addClass("rouge"); **// pour ajouter une classe CSS à un élément**
- $("h1").removeClass("rouge"); **// pour retirer une classe CSS à un élément**

JavaScript



Autres fonctions

- $("#bouton").text(‘changer le label’);
- Var contenu=$("#bouton").text();
- $("#zone").html("bonjour");
- $("#champ").val("mon texte");
- $("input:first").attr("placeholder","Insérer votre nom");



Exemple d'utilisation

Déterminer si une checkbox est cochée

```
If ($('#total').attr('checked')) {
        //Traitement si cochée
    }
    else {
        //Traitement si non cochée
    }
```



Evènements jQuery

Lancement au chargement de la page

- Pour ne lancer un script que lorsque l'on est sûr que l'intégralité du DOM a été chargée. jQuery offre une méthode plus souple, à l'aide de la méthode **ready** :

$(document).**ready**(*GestionnaireALancer*) ;

- On peut ainsi écrire :

$(document).**ready**(function(){...}) ;

Ou bien :

$(document).**ready**(Gestionnaire) ;
function Gestionnaire(evt){...}

- En général une écriture jQuery suit le syntaxe suivant: **$(selecteur).action();**
- Il suffit de placer cette ligne de code entre les balises <script> et </script> dans l'entête du document HTML.



Evènements jQuery

- click() **// au clic de la souris**
- dblclick() **// au double clic de la souris**
- mouseenter() **// lorsque la souris entre dans un espace alloué à un élément**
- mouseleave() **// au moment où la souris quitte un espace alloué à un élément**
- hover() **// lorsque la souris entre dans un espace alloué à un élément et le quitte**
- mousedown() **// au moment où l'on clic et maintient le clic de la souris**
- mouseup() **// au moment où l'on relâche le clic de la souris**
- focus() **// lorsque l'on clic sur un champ input**
- blur() **// lorsque l'on clic en dehors d'un champ input**



Evènements: Exemple

```
<html>
<head>
<script type="text/javascript"
 src="http://ajax.googleapis.com/ajax/libs/jquery/1/jquery.min.js">
</script>
</head>
<body>
 <h1>Mon Titre</h1>
 <script>
 $("h1").click( function(){alert("Bonjour !"); }
 </script>
</body>
</html>
```



Exemple d’utilisation des évènements

```
<html>
<head>
<script type="text/javascript"
   src="http://ajax.googleapis.com/ajax/libs/jquery/1/jquery.min.js"></script>
</head>
<body>
<h1>Mon Titre</h1>
<button>Clic sur moi</button>
    <script>
        $("button").click(
            function(){$("h1").text("Nouveau titre"));}) ;
    </script>
</body>
</html>
```



Evènements: Gestionnaires d’événements

- JQuery fournit deux manières de définir un gestionnaire d’événement :
  - soit en indiquant le nom du gestionnaire:

```
$("p").click(Gestionnaire) ;

function Gestionnaire(evt){
  alert("Ceci est un paragraphe");
}
```

  - soit en codant directement le gestionnaire, par exemple…

```
$("p").click(function(){
alert("Ceci est un paragraphe");
}) ;
```



Evènements: Exercice

- Insérer plusieurs paragraphes dans une page html
- Affecter aux paragraphes le gestionnaire clickP associé au clic. Cette fonction change la couleur de l'élément cliqué en rouge.



Correction d’exercice: Premiers effets

```
<p id="p1">paragraphe 1</p>
<p id="p2">paragraphe 2</p>
<script>
$("#p1").click(clickP);
$("#p2").click(clickP);
function clickP(evt){
//let id= $(this).attr('id');
let id=this.id;
$("#"+id).css("color","red");
}
</script>
```



Quelques effets

- **show()** et **hide()** permettent respectivement de montrer et cacher des éléments. Par exemple, $("p").hide() cache tous les paragraphes du document.
- **show(vitesse)** et **hide(vitesse)** permettent respectivement de montrer et cacher des éléments avec une certaine vitesse. Cette vitesse est indiquée par des mots-clefs ("slow","normal" ou "fast") ou le nombre de millisecondes que doit durer l'animation.
- **toggle()** et **toggle(vitesse)** permettent de basculer d'un mode d'affichage à un autre (un élément caché devient visible, ou un élément visible devient caché).
- **slideDown()** et **slideUp()** permettent de faire apparaître (respectivement disparaître) un élément à la manière d'un store se déroulant ou s'enroulant.
- **slideToggle()** permet de basculer d'un mode d'affichage à un autre.
- **fadeIn(vitesse)** et **fadeOut(vitesse)** permettent de faire progressivement apparaître (ou disparaître) un élément en jouant sur sa transparence.



Exercice: Effets

- Insérer une image et un bouton dans une page html
- Affecter au bouton le gestionnaire « apparaitre » associé au clic. Cette fonction fait apparaître ou disparaitre l'image, le contenu du bouton est modifié selon la situation.



Correction d’exercice: Effets

```
<script src="js/jquery.js"></script>
<img src="images/surfeur.jpg" alt="image"/><br>
<button>disparaitre</button>
<script>
$("button").click(apparaitre);
function apparaitre(evt){
if($("button").text()=="disparaitre"){
$("img").slideUp(2000);
$("button").text("apparaitre");
}else{
$("img").slideDown(2000);
$("button").text("disparaitre");}}
</script>
```